Aflevering 3

Composite Design Pattern

Gruppe 4

I4SWD

16/04-2018

|  |  |  |
| --- | --- | --- |
| Navn: | Studienr.: | auID |
| Daniel Hansen | 201601915 | 565631 |
| Fatima Kodro | 201609565 | 553932 |
| Martin Andersen | 201605036 | 565118 |
| Søren Bech | 201604784 | 568435 |

Indhold

[Indledning 3](#_Toc511383032)

[Struktur og dynamik 3](#_Toc511383033)

[Diagrammer 3](#_Toc511383034)

[Eksempel 3](#_Toc511383035)

[Analyse og konklusion 3](#_Toc511383036)

# Indledning

Som pattern er der blevet valgt composite pattern, der er meget god til at håndtere data i en træstruktur. Composite pattern er derfor god at bruge hvis der ønskes at arbejde med komponenter, der er indenunder andre komponenter.

Som eksempel er der valgt at arbejde med et rang-system over militæret i flåden, hvor et element kan defineres som at være en del af et andet element eller underordnede. Eksemplet består af leaves og composites, hvor leaf i dette tilfælde er den rang, som ikke kan have nogle underordnede. Denne rang er ”Sailor”. Composites er derimod dem, der har mulighed for at have en eller flere underordnede. Da der er flere funktioner der kan bruges i både leaves og composite, samt er der funktioner ud over disse, bruges en component klasse, som beskriver alle funktioner, både dem der er fælles, samt dem, der virker for leaves og composites.

Med Composite pattern er det nemt at traversere alle elementer i træstrukturen, samt er det muligt at traversere over specifikke dele, fx. Over obersten og alle under ham.  
Composite pattern gør det også muligt at komponere objekter ind i en træ-struktur for at repræsentere hierarkier.

# Struktur og dynamik

For at løse opgaven, er der taget udgangspunkt i SOLID:

S – Single Responsibility er ikke blevet brugt, da det giver mest mening at have en overordnet klasse for alle composites. Dette medfører at composite-klassen har flere forskellige kommandoer, der kan udføres og derfor er der ikke single responsibility.

O – Open-Closed princippet er ikke brugt, da hvis der skal ændres noget i enten Leaf- eller Compositeklassen, skal dette også ændres i Componentklassen

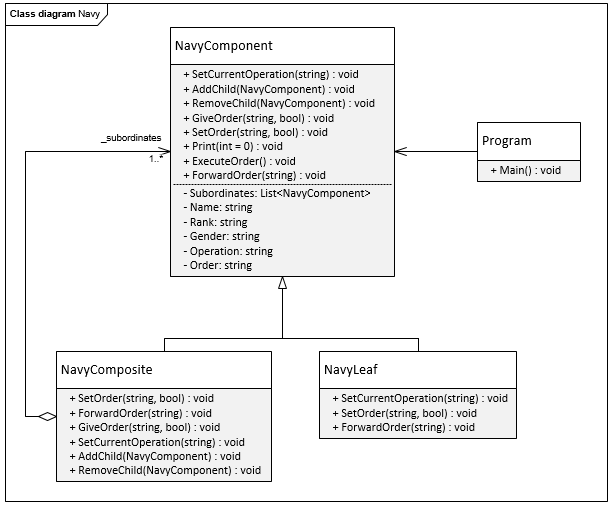
L – Liskov substitution princippet bruges i projektet, da det er muligt at substituere alle leaves og composites. Dette kan fx gøres ved at demote en person.

I – Interface segregation princippet bruges ikke, da der ikke var brug for at lave interfaces til klasserne.

D - Dependency inversion princippet bruges, da der er implementeret en Component abstract klasse, som Leaf og Composite er afhængig af, som er konkrete klasser.

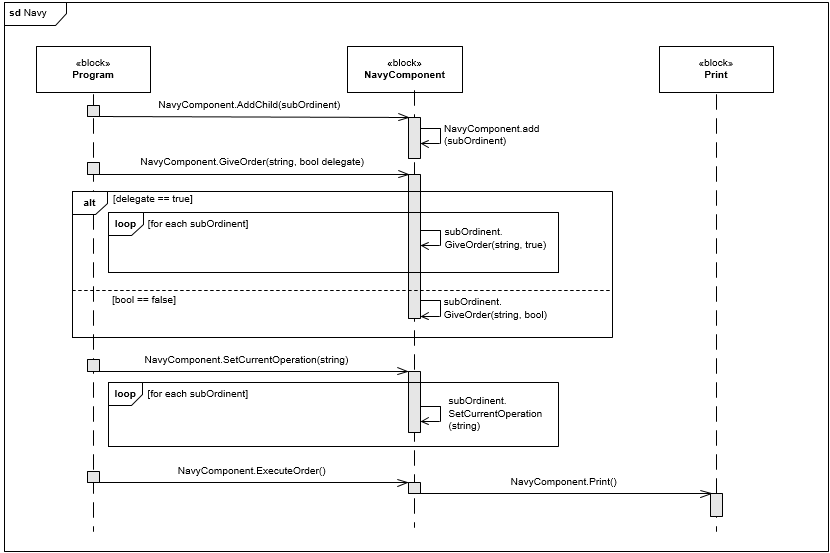
## Diagrammer

På Figur 1, ses et klassediagram over Navy, hvor det ses at NavyComposite og NavyLeaf arver fra NavyComponent. Det er som beskrevet tidligere, fordi Component er en overordnet klasse over de andre. NavyLeaf beskriver den rang som er nederst i systemet, dvs. de kan ikke have nogen underordnede. Derfor har de kun mulighed for at få ordrer og operationer. Composites beskriver de overordnede rang, så derfor har de flere kommandoer, fx at give ordrer og operationer ud til andre.



Figur : Klassediagram over Navy

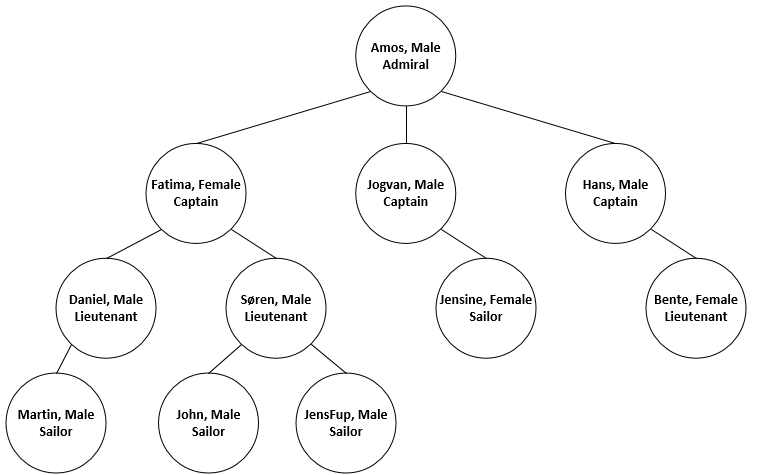
På Figur 2 ses et sekvensdiagram over Navy. Her vises et scenarie af hvilke funktioner, der bliver kaldt i programmet. Der bruges kun NavyComponent, da den repræsenterer både Leaf og Composite. Der vises til at starte med, at der bliver tilføjet en underordnede til en component vha. AddChild-metoden. Herefter gives der en ordrer til denne underordnede. Når der gives en ordre, kan parameteren ”delegate” sættes til enten true eller false. Hvis den sættes til true, vil ordren blive sendt videre ned i hierarkiet, dvs. den bliver sendt ned til alle leaves for den valgte composite. Hvis ”delegate” sættes til false, er det kun de personer der er lige under den valgte composite der får ordren. Herefter bliver der kaldt SetCurrentOperation(). Når dette bliver kaldt, bliver der udført en operation til en composite og alle dens underordnede. Til sidst bliver der kaldt ExecuteOrder, som udføre en components ordre, hvis der er tildelt en til personen.



Figur : Sekvensdiagram over Navy

## Eksempel

Et eksempel på den træstruktur, der er blevet brugt til opgaven kan ses på Figur 3, hvor der i hver knude er beskrevet navn, køn og rang for hver person.



Figur 3: Træstruktur over Navy

# Analyse og konklusion

Composite er godt at bruge, hvis man har en træstruktur som man vil kunne arbejde med. Det er dog (mere eller mindre) det eneste som Composite er god til. Derfor er det ikke en god pattern at bruge, hvis vi har andet data at arbejde på.

Composite pattern kan sammenlignes med Decorator pattern, hvor at forskellen på disse patterns er, at klasserne i Composite arver fra hinanden, dvs. hvis noget kaldes nede i et barn, vil dette blive arvet fra dens forældre og derefter overrides den. I Decorator pattern, vil det først blive kaldt oppe i forælderen, og derefter blive kaldt i barnet. Vi synes at Composite giver mest mening da der her ikke skal eksekveres for meget ved kørsel af programmet.

**1 Formalia**

1. All requested files are submitted.

2. The naming of folders, files and visual-studio solution have been done in accordance with the specifications.

3. The size of the video files are in accordance with the specifications.

**2 Description (Structure and Dynamics of the pattern)**

1. Does the group demonstrate full understanding of the subject?

2. Do they cover all aspects of the chosen pattern and use correct and relevant concepts from the course, e.g. referencing design principles (SOLID)?

3. Do they present a good and understandable example of how the pattern can be used?

4. Do they make good use of relevant diagrams and are they correct, both in notation and contents?

5. Is it clear how you should use the pattern and what the possible variations are (if any)?

**3 Analysis and conclusions**

1. Does the group present a sound analysis of the patterns usage, when it is good and bad to use it?

2. Do they compare it to other relevant design patterns and explain similarities and differences?

3. Is it clear when you should use the pattern and why?

**4 Presentation material**

1. Is the presentation (both slides and presenters) clear, precise and understandable?

2. Is the presentation on time, and do they use the time wisely?

3. Is the demonstration video clear, precise and understandable?

4. Are references put to material (both code and written) on which the solution to the assignment is based?

**5 Solution / demos**

1. Is a prototype solution presented to highlight the key concepts of the pattern?

2. Does the code example(s) compile and run without errors?

3. Is the naming of classes, variables and methods clear and consistent?

4. Is the code is easy to read and have a proper amount of comments?

5. Does the example(s) demonstrate all core parts of the pattern and relevant variants of its use?